Une fonction qu'un autre chargé peut exécuter

Transformer un script en fonctions avec arguments et stopifnot(), l'évaluation tidy nécessaire pour passer un nom de colonne, et le script en ligne de commande qui s'exécute tel quel sur l'export du trimestre suivant.

Pierre Robentz Cassion

Leçon 8 sur 8

Ce que couvre cette leçon

- Le test de la transmission
- Du script à la fonction
- `stopifnot()` pour ce qui ne peut pas arriver
- Passer un nom de colonne : l'évaluation tidy
- La fonction d'indicateur
- Tester la partie qui calcule
- Le script en ligne de commande
- Consigner ce qui a produit la sortie
- Où ce cours s'arrête

Le test de la transmission

- Une analyse est terminée quand quelqu'un d'autre peut l'exécuter sur l'export du trimestre prochain sans modifier le code.

Du script à la fonction — En R (suite)

```r
# R/clean_register.R

clean_register <- function(muac, gam_mm = 125, sam_mm = 115) {
  stopifnot(is.data.frame(muac))
  required <- c("child_id", "commune", "muac_mm", "oedema")
  missing <- setdiff(required, names(muac))
  if (length(missing) > 0) {
    stop("Colonnes absentes de l'entree : ", paste(missing, collapse = ", "))
  }

  muac |>
    dplyr::mutate(
      # Un PB sous 40 est en centimetres dans un champ en millimetres. Applique
      # seulement sous un seuil qu'aucune mesure plausible n'atteint.
      muac_mm = dplyr::if_else(muac_mm < 40, muac_mm * 10L, muac_mm),
      muac_mm = dplyr::if_else(dplyr::between(muac_mm, 80L, 220L), muac_mm, NA_integer_),
```

Du script à la fonction — En R (suite)

```r
      oedema  = dplyr::recode(
        tolower(trimws(oedema)),
        "true" = TRUE, "y" = TRUE, "yes" = TRUE,
        "false" = FALSE, "n" = FALSE, "no" = FALSE,
        .default = NA
      ),
      assessed = !is.na(muac_mm) | !is.na(oedema),
      gam = assessed & (muac_mm < gam_mm | oedema),
      sam = assessed & (muac_mm < sam_mm | oedema)
    )
}
```

Du script à la fonction

- Elle prend un tableau de données et en renvoie un — Aucune lecture de fichier, aucune écriture, aucun «-
- Les seuils sont des arguments avec valeurs par défaut — Un seuil qu'on ne change qu'en éditant une ligne finira par…
- Elle valide son entrée — Un `stop()` nommant les colonnes absentes vaut mieux qu'un `NULL` se propageant dans un mutate…
- Rien n'est groupé en sortie — Un tableau groupé qui s'échappe d'une fonction est l'erreur de la leçon sur le…

`stopifnot()` pour ce qui ne peut pas arriver — En R

```
stopifnot(
  "chaque ligne exige un identifiant" = !any(is.na(muac$child_id)),
  "le PB doit etre plausible apres nettoyage" =
    all(is.na(muac$muac_mm) | dplyr::between(muac$muac_mm, 80, 220))
)
```

Passer un nom de colonne : l'évaluation tidy — En R

```r
# Ne fonctionne pas.
rate_by <- function(df, group_col) {
  df |> dplyr::summarise(n = dplyr::n(), .by = group_col)
}

rate_by(muac, commune)
#> Error: object 'commune' not found
```

Passer un nom de colonne : l'évaluation tidy — En R

```r
rate_by <- function(df, group_col) {
  df |> dplyr::summarise(n = dplyr::n(), .by = {{ group_col }})
}

rate_by(muac, commune)
```

Passer un nom de colonne : l'évaluation tidy — En R

```r
rate_by_name <- function(df, group_col) {
  df |> dplyr::summarise(n = dplyr::n(), .by = dplyr::all_of(group_col))
}

rate_by_name(muac, "commune")
```

Passer un nom de colonne : l'évaluation tidy — En R

```r
count_as <- function(df, group_col, out_name) {
  df |> dplyr::summarise("{out_name}" := dplyr::n(), .by = {{ group_col }})
}
```

La fonction d'indicateur — En R

```r
indicator_table <- function(muac, gam_mm = 125, sam_mm = 115) {
  clean_register(muac, gam_mm, sam_mm) |>
    dplyr::summarise(
      screened  = dplyr::n(),
      assessed  = sum(assessed, na.rm = TRUE),
      gam_cases = sum(gam, na.rm = TRUE),
      sam_cases = sum(sam, na.rm = TRUE),
      .by = commune
    ) |>
    dplyr::mutate(
      assessment_rate = round(assessed / screened, 3),
      gam_rate = round(gam_cases / assessed, 3),
      sam_rate = round(sam_cases / assessed, 3)
    ) |>
    dplyr::arrange(dplyr::desc(gam_rate))
}
```

Tester la partie qui calcule — En R

```r
# tests/testthat/test-indicator-table.R
test_that("le denominateur exclut les enfants non mesures", {
  muac <- tibble::tibble(
    child_id = c("a", "b", "c"),
    commune  = "X",
    muac_mm  = c(110L, 130L, NA_integer_),
    oedema   = c("false", "false", NA_character_)
  )

  out <- indicator_table(muac)

  expect_equal(out$screened, 3)
  expect_equal(out$assessed, 2)     # l'enfant non mesure n'est pas au denominateur
  expect_equal(out$gam_rate, 0.5)
})
```

Le script en ligne de commande — En R (suite)

```r
#!/usr/bin/env Rscript
# scripts/indicator_table.R

suppressPackageStartupMessages({
  library(optparse)
  library(readr)
})

source(here::here("R", "clean_register.R"))
source(here::here("R", "indicator_table.R"))

option_list <- list(
  make_option("--input", type = "character", help = "CSV du registre de depistage."),
  make_option("--output", type = "character", help = "Repertoire des tableaux."),
  make_option("--gam-threshold", type = "integer", default = 125L,
              help = "Seuil de PB en mm pour la MAG [defaut %default].")
```

Le script en ligne de commande — En R (suite)

```
)

opt <- parse_args(OptionParser(option_list = option_list))
if (is.null(opt$input) || is.null(opt$output)) {
  stop("--input et --output sont requis.")
}

muac <- read_register(opt$input)
message("Lu ", nrow(muac), " lignes depuis ", opt$input)

table <- indicator_table(muac, gam_mm = opt$`gam-threshold`)

thin <- table$commune[table$assessment_rate < 0.8]
if (length(thin) > 0) {
  warning("Taux d'evaluation sous le plancher : ", paste(thin, collapse = ", "))
}
```

Le script en ligne de commande — En R (suite)

```
dir.create(opt$output, recursive = TRUE, showWarnings = FALSE)
destination <- file.path(opt$output, "gam_by_commune.csv")
write_csv(table, destination)
message("Ecrit ", destination, " (", nrow(table), " communes)")
```

Le script en ligne de commande — Shell

```
Rscript scripts/indicator_table.R \
  --input data/raw/muac-screening-artibonite-2024.v1.csv \
  --output outputs/tables
```

Consigner ce qui a produit la sortie — En R

```r
run_metadata <- function(opt) {
  list(
    generated_at  = format(Sys.time(), "%Y-%m-%dT%H:%M:%S%z"),
    input         = opt$input,
    gam_threshold = opt$`gam-threshold`,
    r_version     = as.character(getRversion()),
    dplyr         = as.character(packageVersion("dplyr"))
  )
}

jsonlite::write_json(run_metadata(opt), file.path(opt$output, "run.json"),
                     auto_unbox = TRUE, pretty = TRUE)
```

Consigner ce qui a produit la sortie — Shell

```
rm -rf outputs/
Rscript scripts/indicator_table.R --input data/raw/muac.csv --output outputs/tables
git status
```

Où ce cours s'arrête

- Vous savez construire un projet qui se rouvre un an plus tard avec ses versions de paquets consignées, lire n'importe quel export sans le corrompre, conserver les étiquettes qu'un fichier d'enquête transporte, mettre les catégories dans l'ordre qu'exige un rapport, agréger vers un numérateur et un dénominateur issus d'un seul appel, restructurer un groupe répété aplati, et remettre le résultat à quelqu'un d'autre sous forme de script plutôt que de service rendu.

Lire la leçon complète, avec le code exécutable

https://data-analysis.cassion.dev/fr/cours/r-for-programme-data/r-08-functions